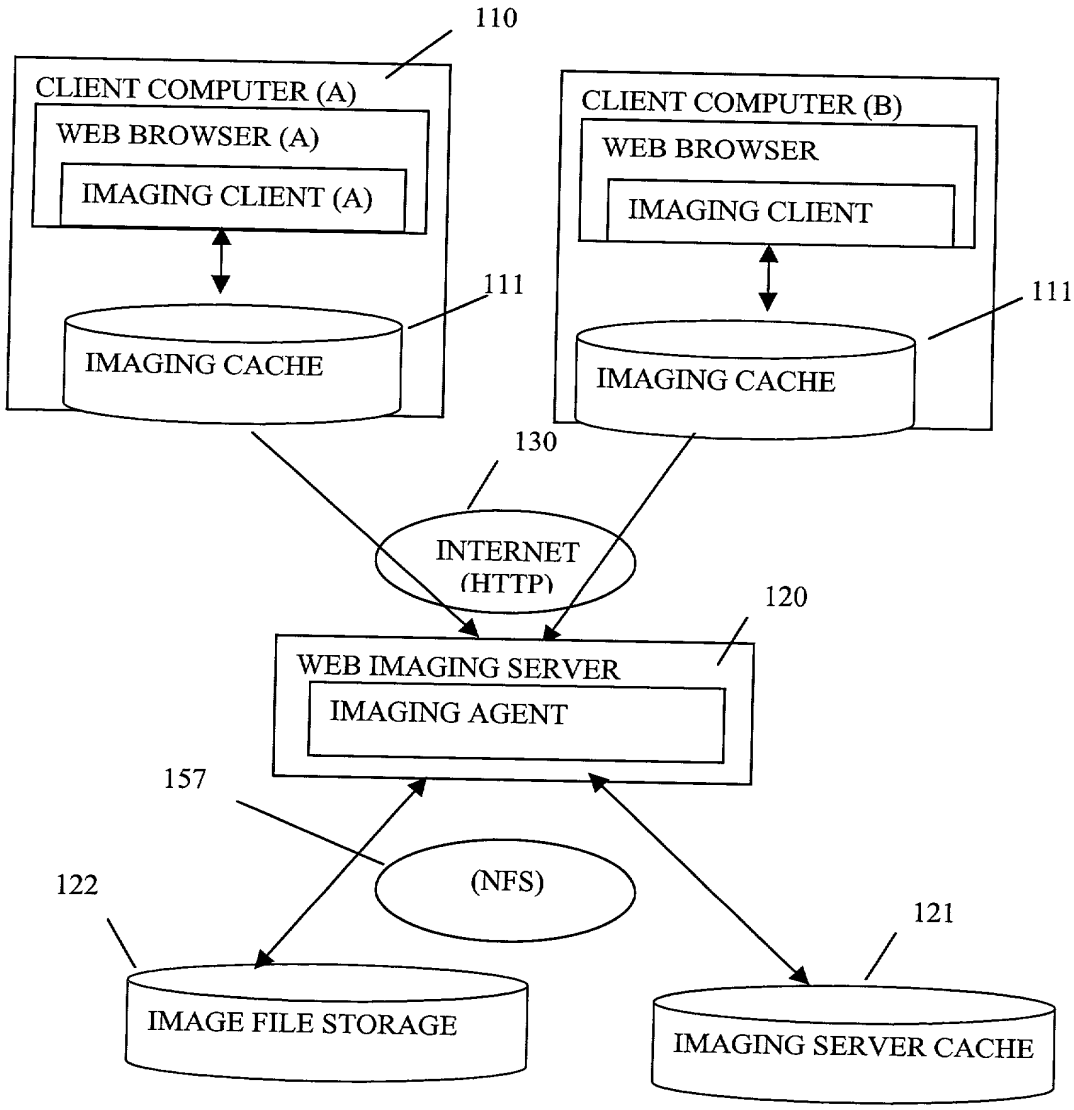

110
CLIENT COMPUTER (A)
WEB BROWSER (A)
IMAGING CLIENT (A)
111
IMAGING CACHE
CLIENT COMPUTER (B)
WEB BROWSER
IMAGING CLIENT
111
IMAGING CACHE
130
INTERNET
(HTTP)
120
WEB IMAGING SERVER
IMAGING AGENT
157
(NFS)
122
IMAGE FILE STORAGE
121
IMAGING SERVER CACHE

Fig. 1

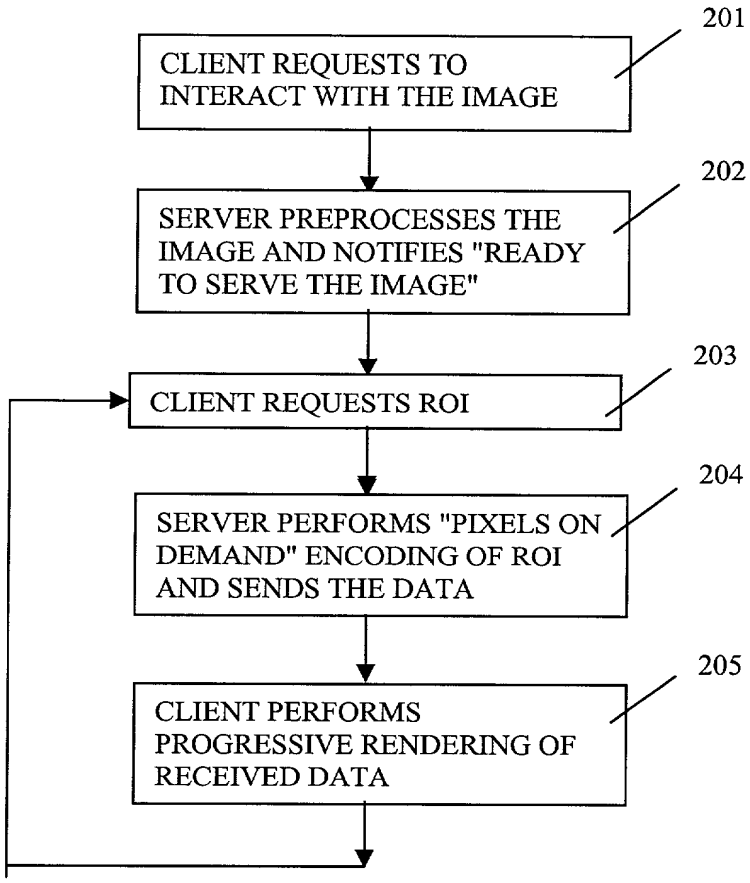

201
CLIENT REQUESTS TO INTERACT WITH THE IMAGE
202
SERVER PREPROCESSES THE IMAGE AND NOTIFIES "READY TO SERVE THE IMAGE"
203
CLIENT REQUESTS ROI
204
SERVER PERFORMS "PIXELS ON DEMAND" ENCODING OF ROI AND SENDS THE DATA
205
CLIENT PERFORMS PROGRESSIVE RENDERING OF RECEIVED DATA

Fig. 2

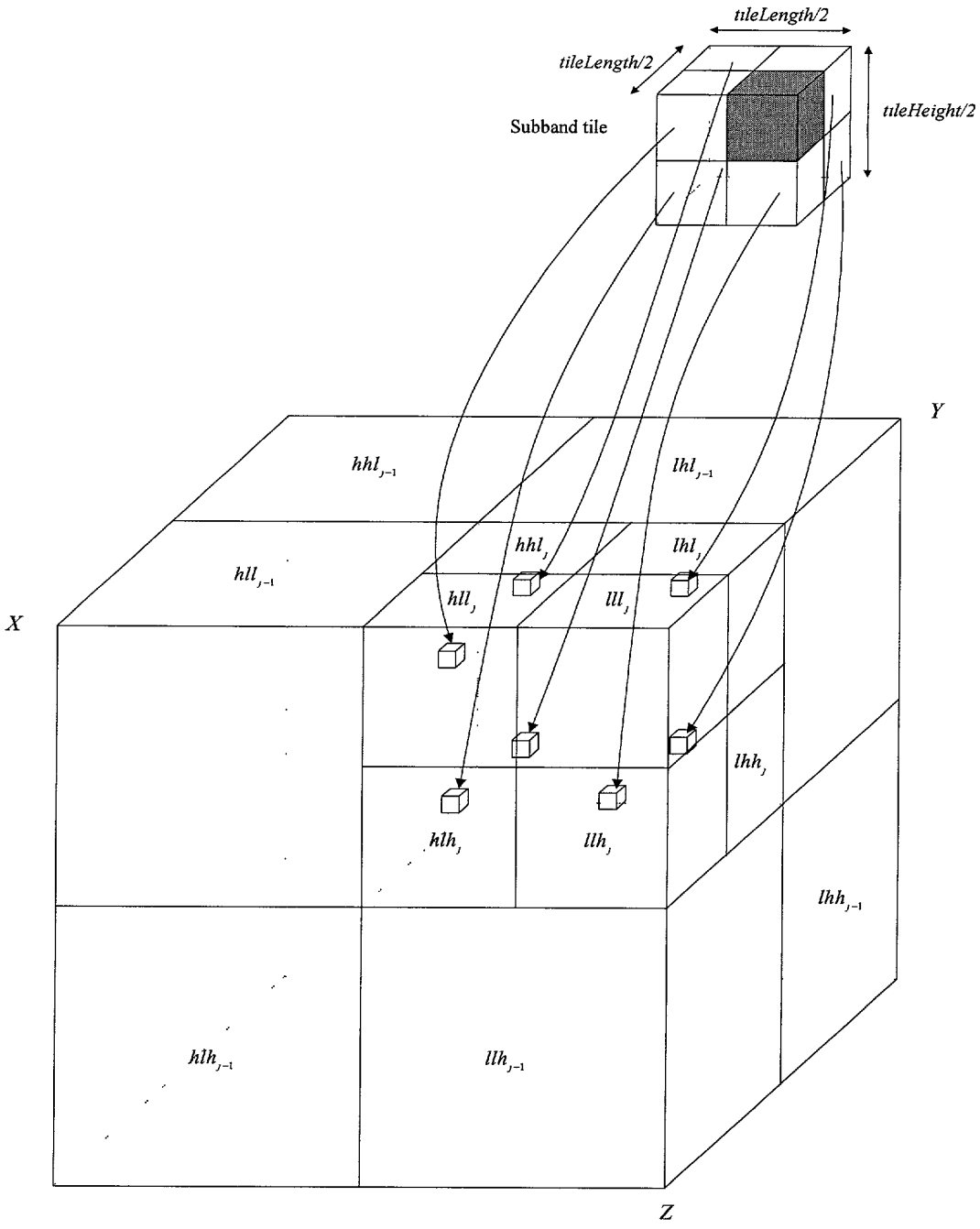

tileLength/2
tileLength/2
tileHeight/2
Subband tile
Y
X
Z
hhl_{j-1}
lhl_{j-1}
hll_{j-1}
hhl_j
lhl_j
hll_j
lll_j
lhh_j
hlh_j
llh_j
lhh_{j-1}
hlh_{j-1}
llh_{j-1}

Fig. 3

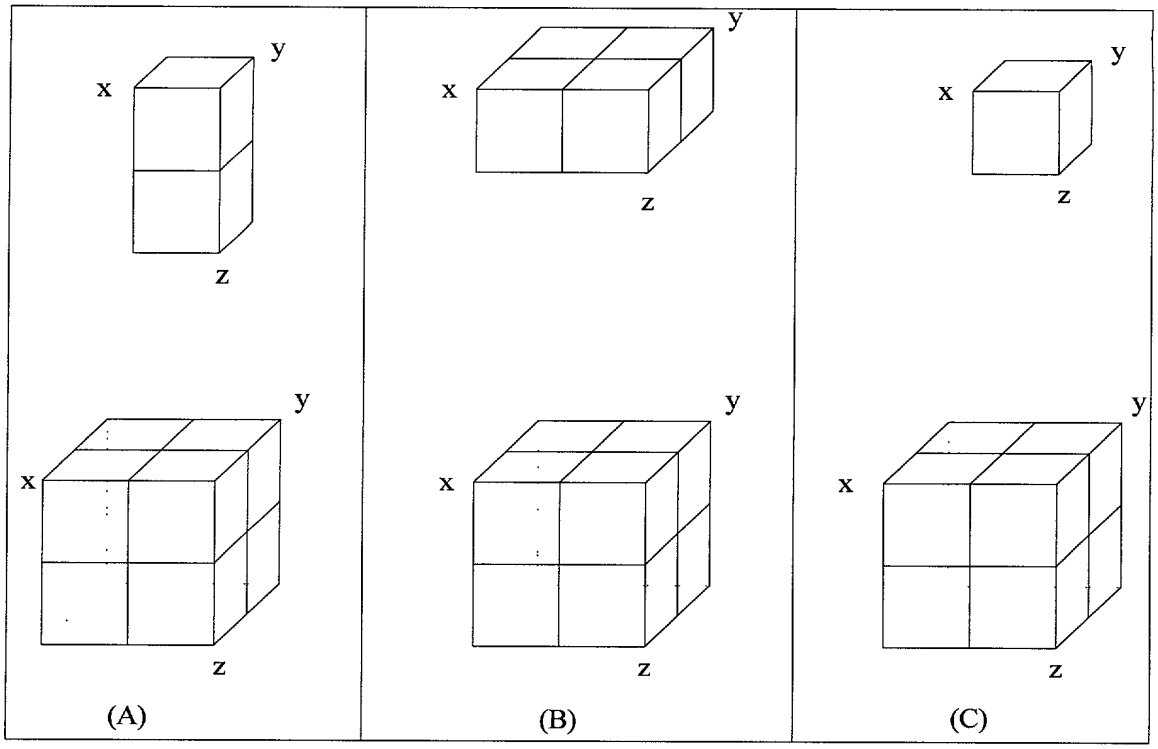


Fig. 4

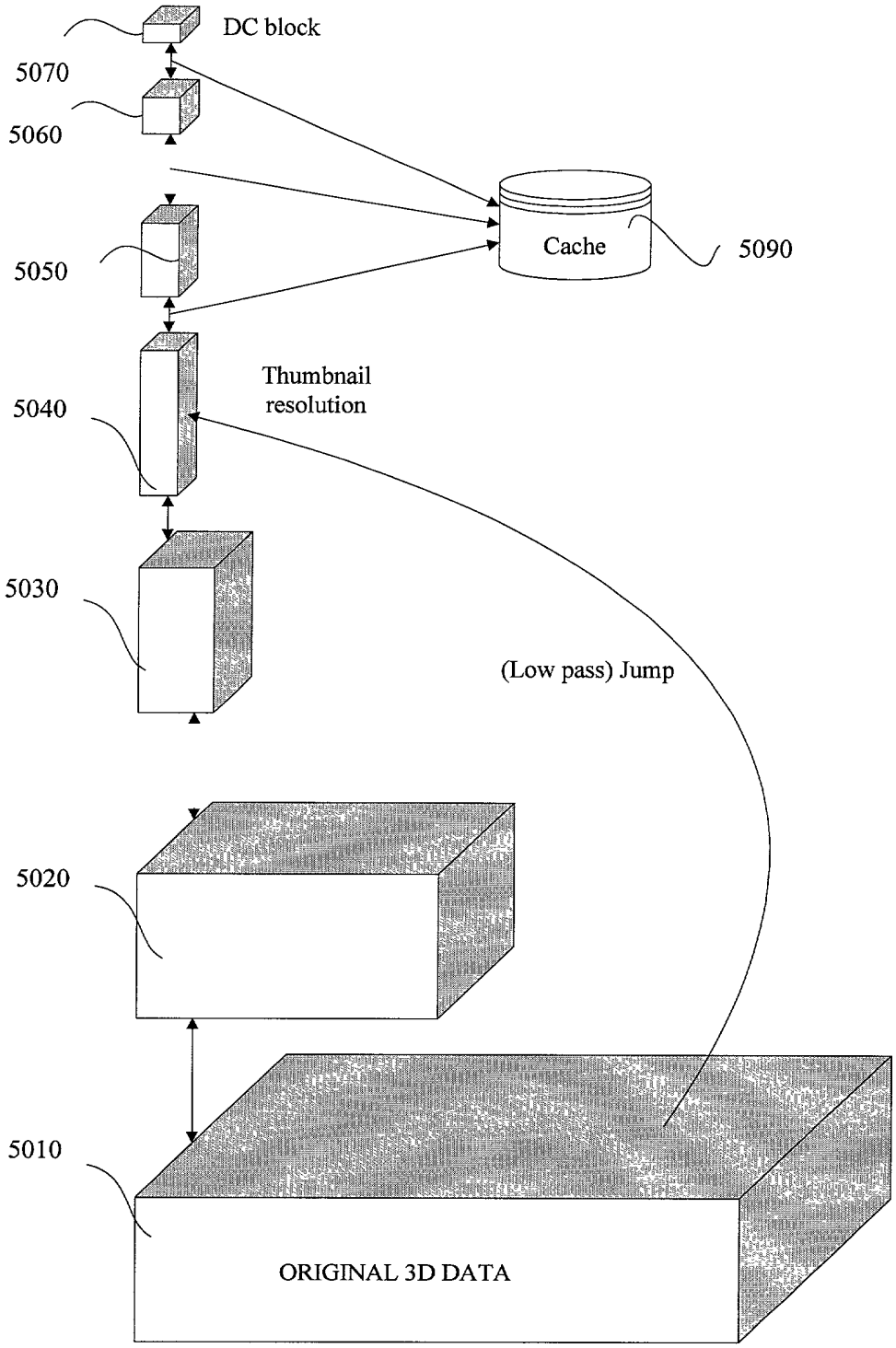

DC block
5070
5060
Cache
5090
5050
Thumbnail resolution
5040
5030
(Low pass) Jump
5020
5010
ORIGINAL 3D DATA

Fig. 5

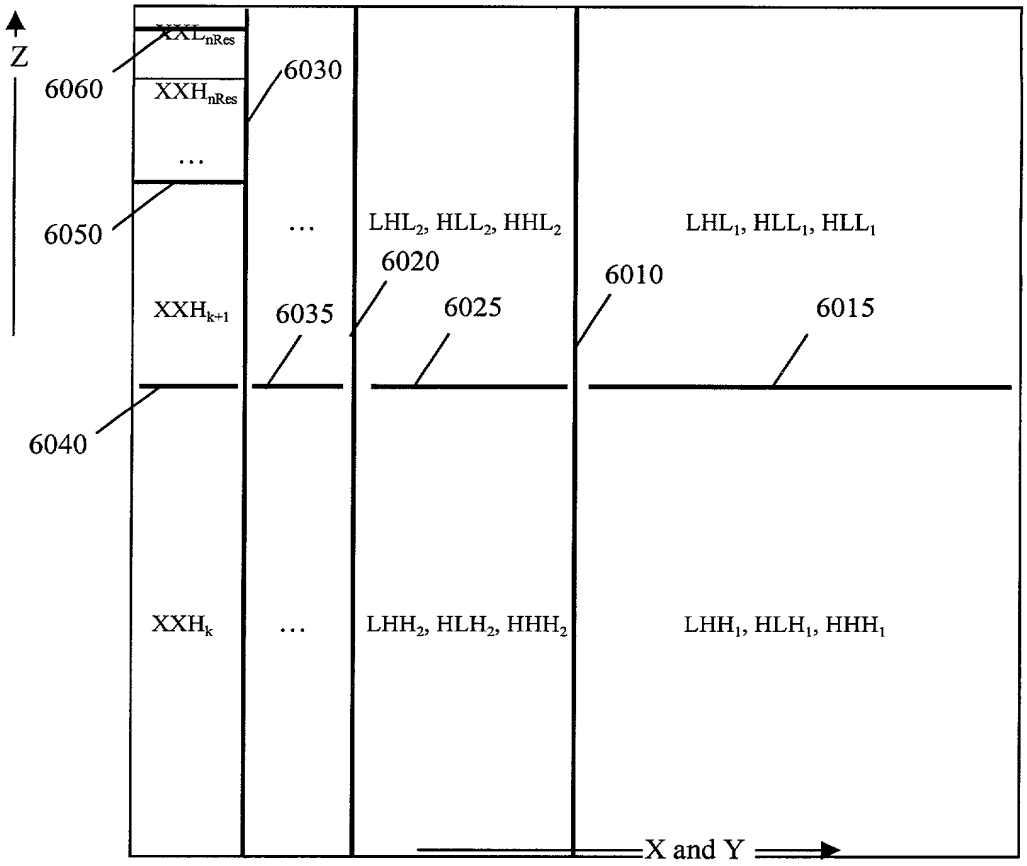

Z
6060
XXL_{nRes}
XXH_{nRes}
...
6030
6050
...
LHL_2, HLL_2, HHL_2
LHL_1, HLL_1, HLL_1
6020
6010
XXH_{k+1}
6035
6025
6015
6040
XXH_k
...
LHH_2, HLH_2, HHH_2
LHH_1, HLH_1, HHH_1
X and Y

Fig. 6

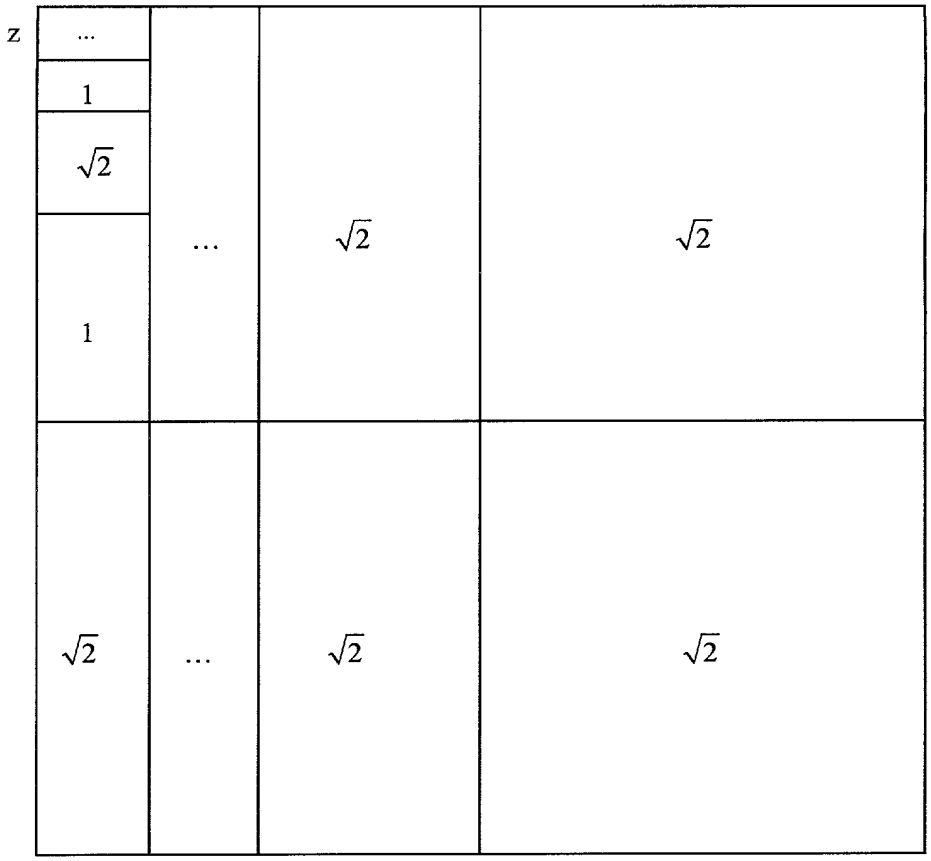
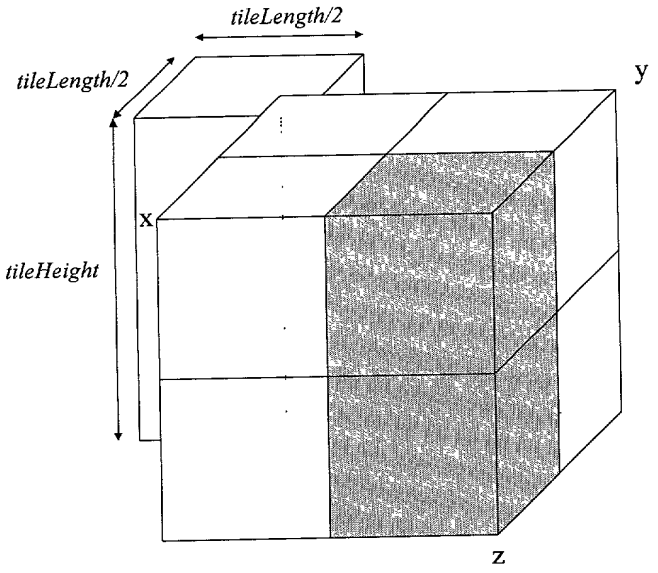
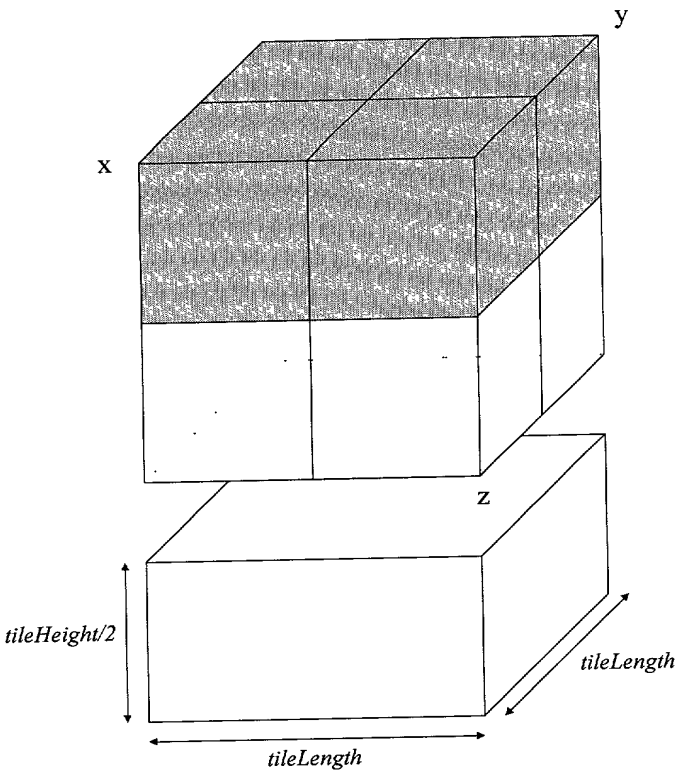


Fig. 7


tileLength/2
tileLength/2
y
x
tileHeight
z

(A)


y
x
z
tileHeight/2
tileLength
tileLength

(B)

Fig. 8

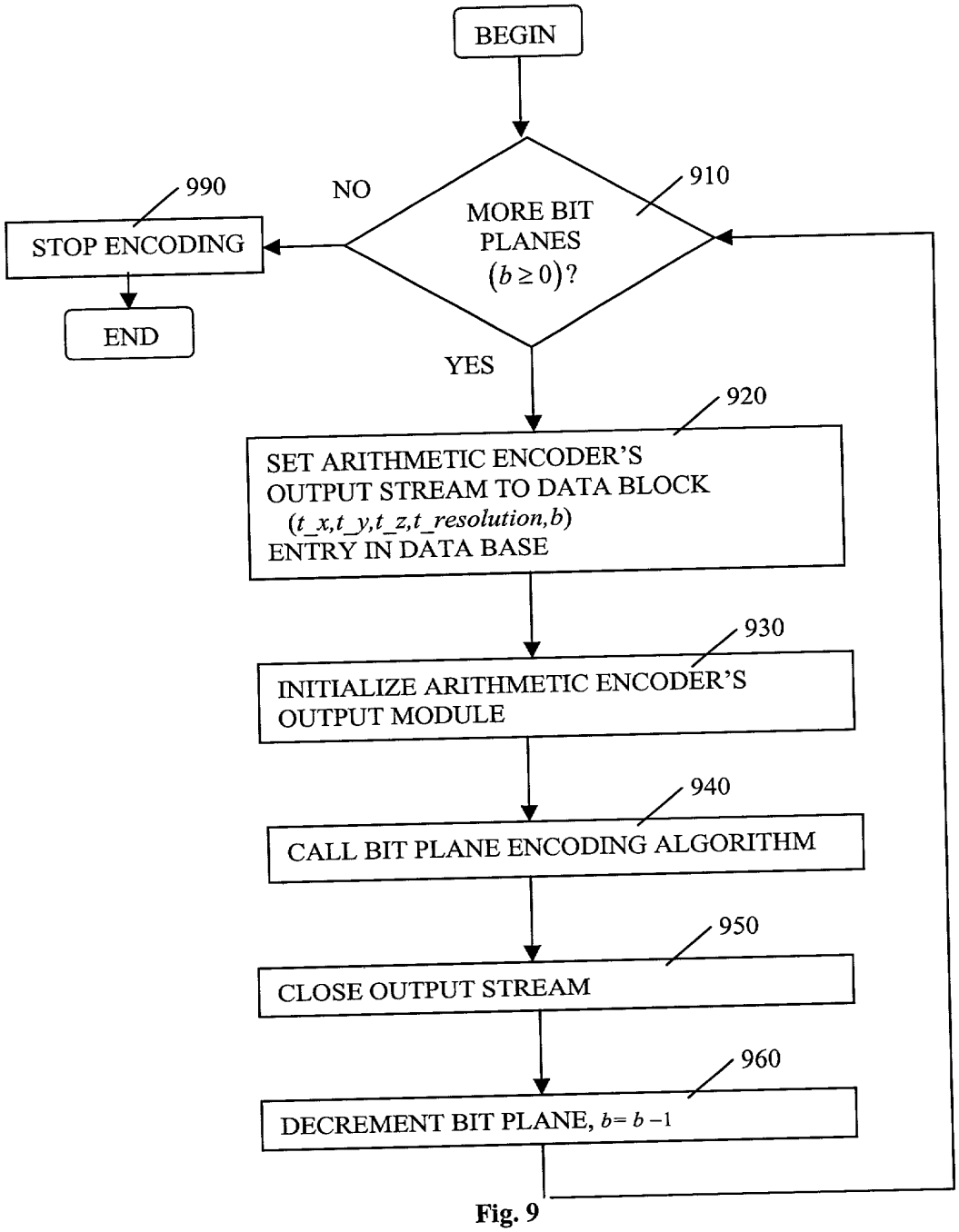

BEGIN
MORE BIT PLANES $(b \geq 0)$?
910
NO
990
STOP ENCODING
END
YES
920
SET ARITHMETIC ENCODER'S OUTPUT STREAM TO DATA BLOCK $(t\_x,t\_y,t\_z,t\_resolution,b)$ ENTRY IN DATA BASE
930
INITIALIZE ARITHMETIC ENCODER'S OUTPUT MODULE
940
CALL BIT PLANE ENCODING ALGORITHM
950
CLOSE OUTPUT STREAM
960
DECREMENT BIT PLANE, $b = b - 1$

Fig. 9

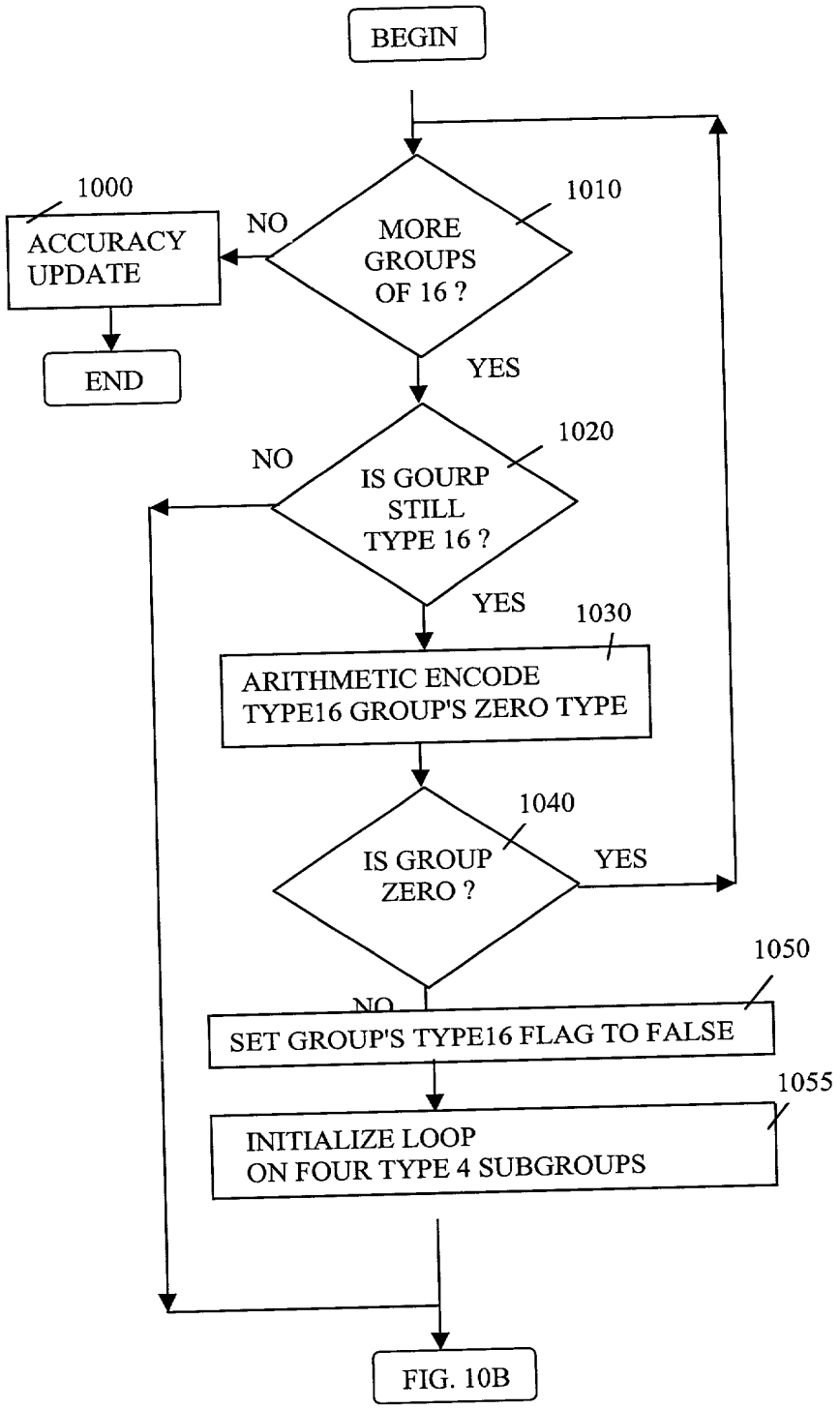

BEGIN
1000
ACCURACY UPDATE
NO
1010
MORE GROUPS OF 16 ?
END
YES
1020
NO
IS GOURP STILL TYPE 16 ?
YES
1030
ARITHMETIC ENCODE TYPE16 GROUP'S ZERO TYPE
1040
IS GROUP ZERO ?
YES
1050
NO
SET GROUP'S TYPE16 FLAG TO FALSE
1055
INITIALIZE LOOP ON FOUR TYPE 4 SUBGROUPS
FIG. 10B

Fig. 10A

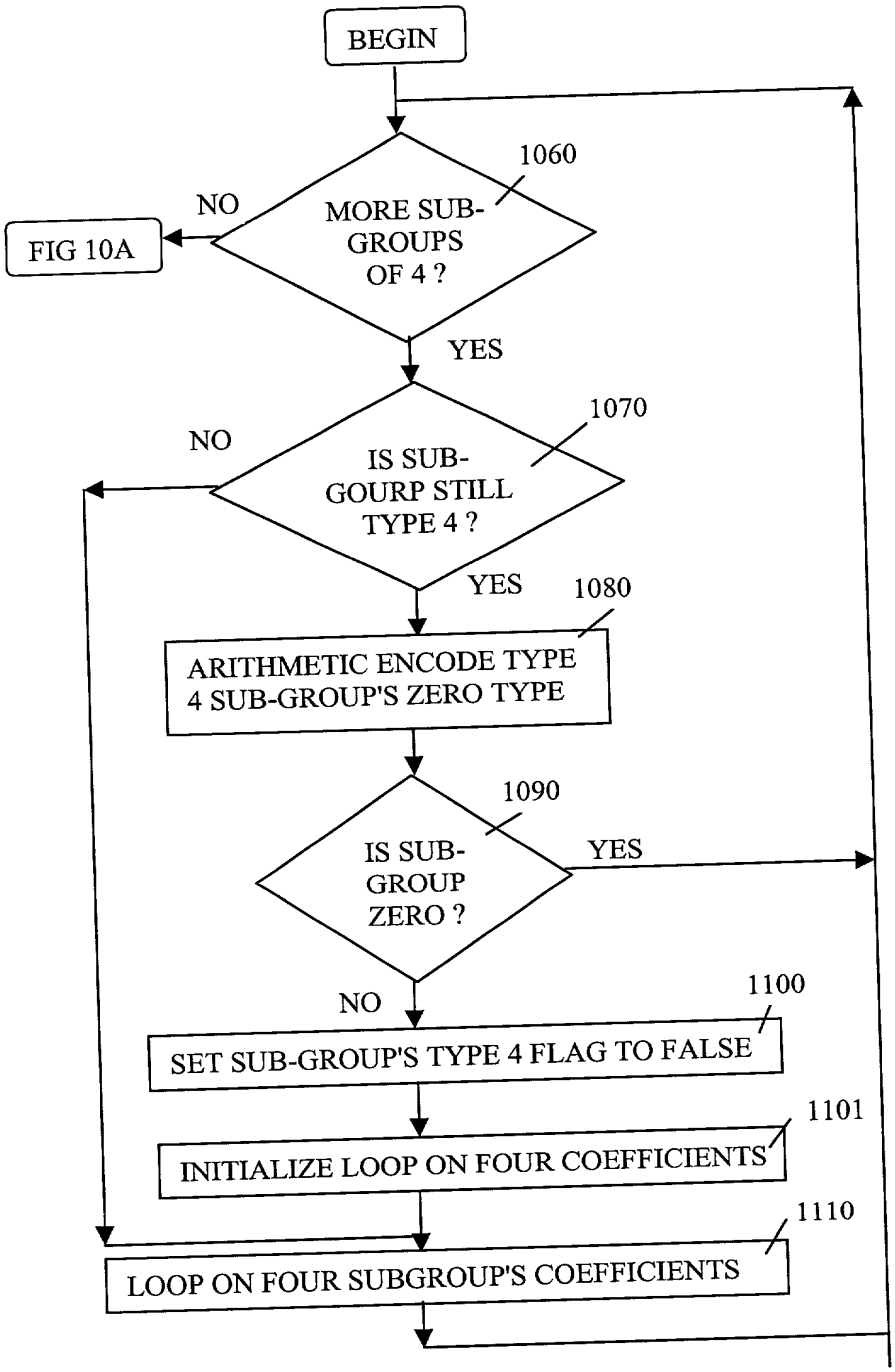

BEGIN
1060
MORE SUB-GROUPS OF 4 ?
NO
FIG 10A
YES
1070
IS SUB-GOURP STILL TYPE 4 ?
NO
YES
1080
ARITHMETIC ENCODE TYPE 4 SUB-GROUP'S ZERO TYPE
1090
IS SUB-GROUP ZERO ?
YES
NO
1100
SET SUB-GROUP'S TYPE 4 FLAG TO FALSE
1101
INITIALIZE LOOP ON FOUR COEFFICIENTS
1110
LOOP ON FOUR SUBGROUP'S COEFFICIENTS

Fig. 10B

```
zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start_model();

while(encoder.getNextGroupOf16()) {
    bool isZero;

    if (encoder.isGroupType16()) {
       isZero = encoder.isZeroGroupOf16();
       arithmetic_encode_symbol(ZeroModel_16,isZero);
       if (isZero)
             continue;
    }

    while (encoder.getNextGroupOf4()) {
       if (encoder.isGroupType4()) {
           if (!encoder.mustbeNoZeroGroup()) {
              isZero = encoder.isZeroGroupOf4();
              arithmetic_encode_symbol(ZeroModel_4,isZero);
              if (isZero)
                    continue;
           }
       }

       while (encoder.getNext_Type1_Coef(isZero)) {
           if (!encoder.mustbeNoZeroCoef())
              arithmetic_encode_symbol(zeroCoefModel,isZero);
           if (!isZero)
              arithmetic_encode_symbol(coefSignModel,encoder.getCoefSign));
       }
    }

    if (!(encoder.isLastBitPlane() && equalBinSetting)) {
       bitModel.start_model();

       int bit;
```

Fig. 11

```
bitModel.startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();
while (encoder.moreCoef()) {                                    1210
      if (encoder.isCoefReported()) {                           1220
            arithmetic_encode_symbol(
                  bitModel,encoder.reportedCoefPrecisionBit());
     } else {
           if ( encoder.isCoefExactZero() )                     1230
                  arithmetic_encode_symbol(zeroCoefModel,true);
           else {
                  arithmetic_encode_symbol(zeroCoefModel,false);
                  arithmetic_encode_symbol(
                        coefSignModel,encoder.getCoefSign());
           }
```

Fig. 12A

```
bitModel.startModel();

for (int z = 0 ; z != HalfBitPlaneZSize ; z++) {
    for (int y = 0 ; y != HalfBitPlaneYSize ; y++) {
        for (int x = 0 ; x != HalfBitPlaneXSize ; x++) {
            arithmetic_encode_symbol(bitModel, coefHalfBit[x][y][z]);
        }
    }
}
```

Fig. 12B

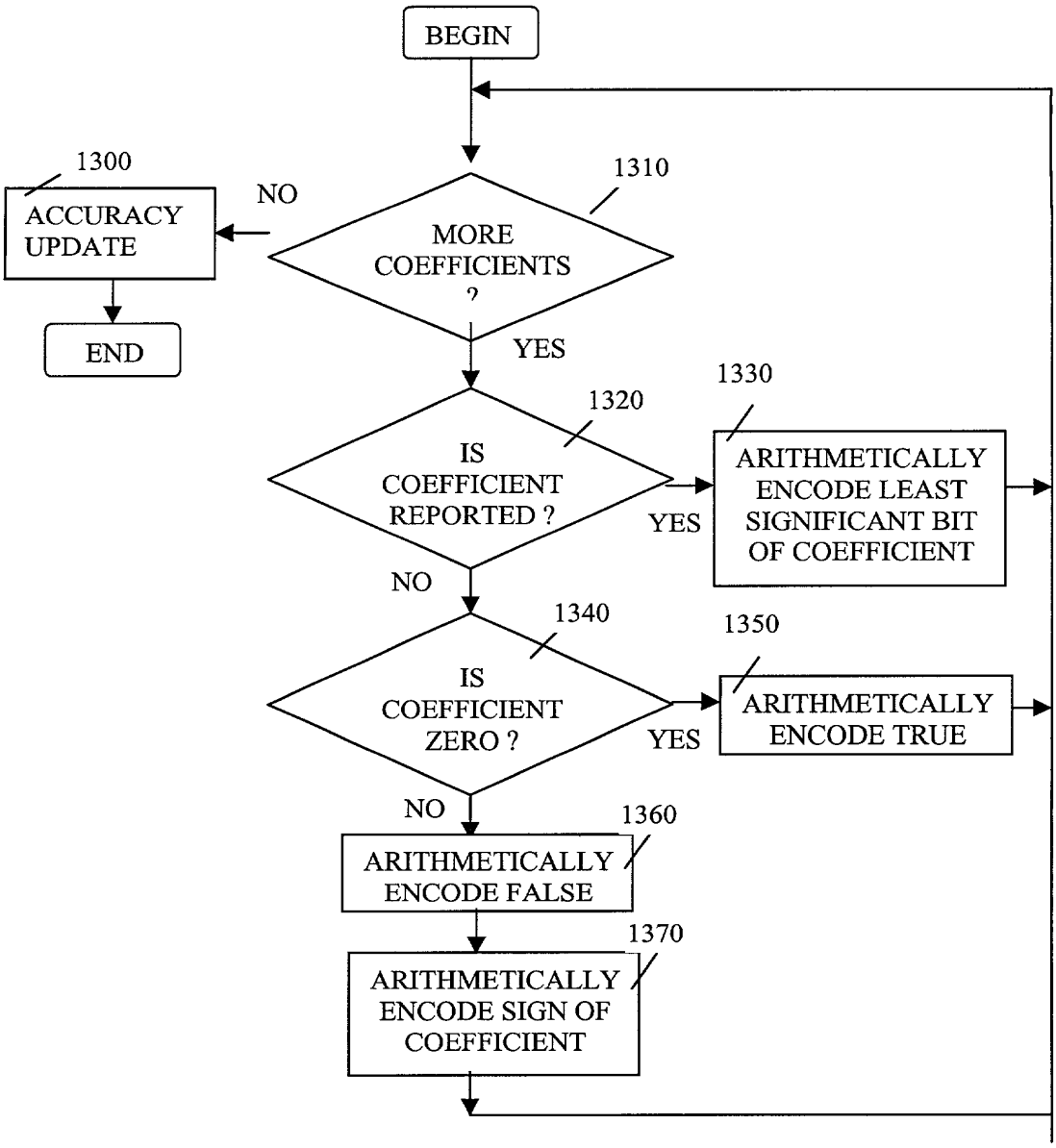

BEGIN
1300
ACCURACY UPDATE
NO
1310
MORE COEFFICIENTS ?
END
YES
1330
1320
IS COEFFICIENT REPORTED ?
ARITHMETICALLY ENCODE LEAST SIGNIFICANT BIT OF COEFFICIENT
YES
NO
1340
1350
IS COEFFICIENT ZERO ?
ARITHMETICALLY ENCODE TRUE
YES
NO
1360
ARITHMETICALLY ENCODE FALSE
1370
ARITHMETICALLY ENCODE SIGN OF COEFFICIENT

Fig. 13

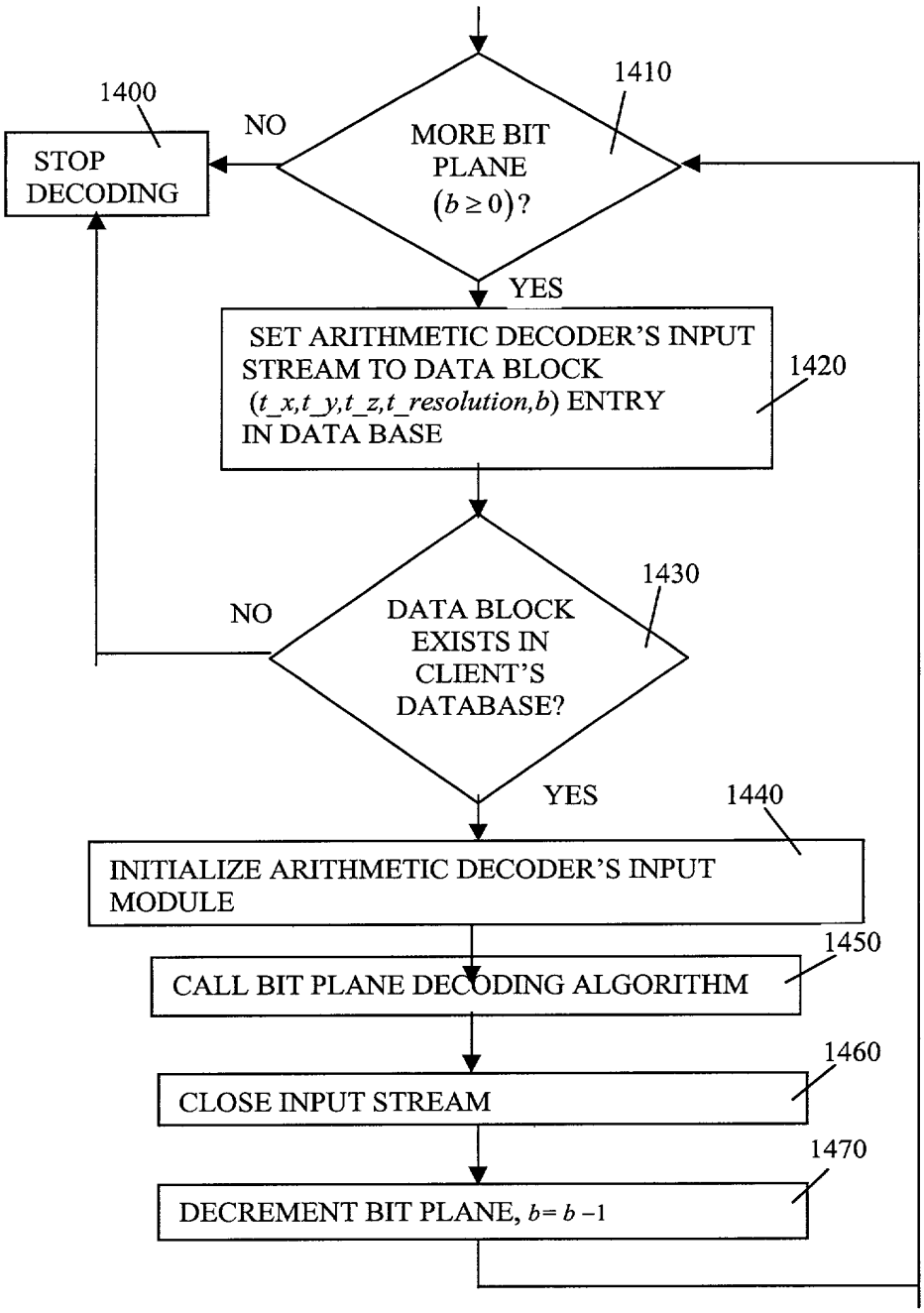

1400
STOP DECODING
NO
1410
MORE BIT PLANE ($b \geq 0$)?
YES
SET ARITHMETIC DECODER'S INPUT STREAM TO DATA BLOCK (*t_x,t_y,t_z,t_resolution,b*) ENTRY IN DATA BASE
1420
1430
NO
DATA BLOCK EXISTS IN CLIENT'S DATABASE?
YES
1440
INITIALIZE ARITHMETIC DECODER'S INPUT MODULE
1450
CALL BIT PLANE DECODING ALGORITHM
1460
CLOSE INPUT STREAM
1470
DECREMENT BIT PLANE, $b = b - 1$

Fig. 14

```
zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start_model();

while(decoder.getNextGroupOf16()) {
    if (decoder.isGroupType16()) {
        if (arithmetic_decode_symbol(zeroModel_16)) {
            decoder.zeroGroupOf16();
            continue;
        }
        else
            decoder.removeZeroGroupOf16();
    }

    while (decoder.getNextGroupOf4()) {
      if (decoder.isGroupType4()) {
        if (!decoder.mustbeNotZeroGroup()) {
              if (arithmetic_decode_symbol(zeroModel_4)) {
                  decoder.zeroGroupOf4();
                  continue;
              }
        }
        decoder.removeZeroGroupOf4();
      }

      while (decoder.getNext_Type1_Coef()) {
        if (decoder.mustbeNotZeroCoef())

decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignModel));
        else if (!arithmetic_decode_symbol(zeroCoefModel))

      decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignModel));
      }
   }
}

if (! (decoder.isLastBitPlane() && equalBinSetting)) {
   bitModel.start_model();
   while(decoder.moreSignificantCoef())
      decoder.setSignificantCoefBit(arithmetic_decode_symbol(bitModel));
}
```

Fig. 15

```
bitModel .startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();

decoder.initializeLSBPlaneCoefScan();                    1610

while (decoder.moreCoef()) {
      if (decoder.isCoefReported()) {
            if(decoder.isSkippedCoef()) {
                  decoder. updateLSB (0);
            }
            else {

decoder.updateLSB(arithmetic_decoder_symbol(bitModel));
            }
      }
      else {
             if(!decoder.isSkippedCoef()) {
                   if (!arithmetic_decoder_symbol(zeroCoefModel))

decoder.setLSB(arithmetic_decoder_symbol(coefSignModel));
             }
```

Fig. 16A

```
bitModel.startModel();

for (int z = 0 ; z != HalfBitPlaneZSize ; z++) {
    for (int y = 0 ; y != HalfBitPlaneYSize ; y++) {
        for (int x = 0 ; x != HalfBitPlaneXSize ; x++) {
            coefHalfBit[x][y][z] = arithmetic_decoder_symbol(bitModel);
        }
    }
}
```

Fig. 16B

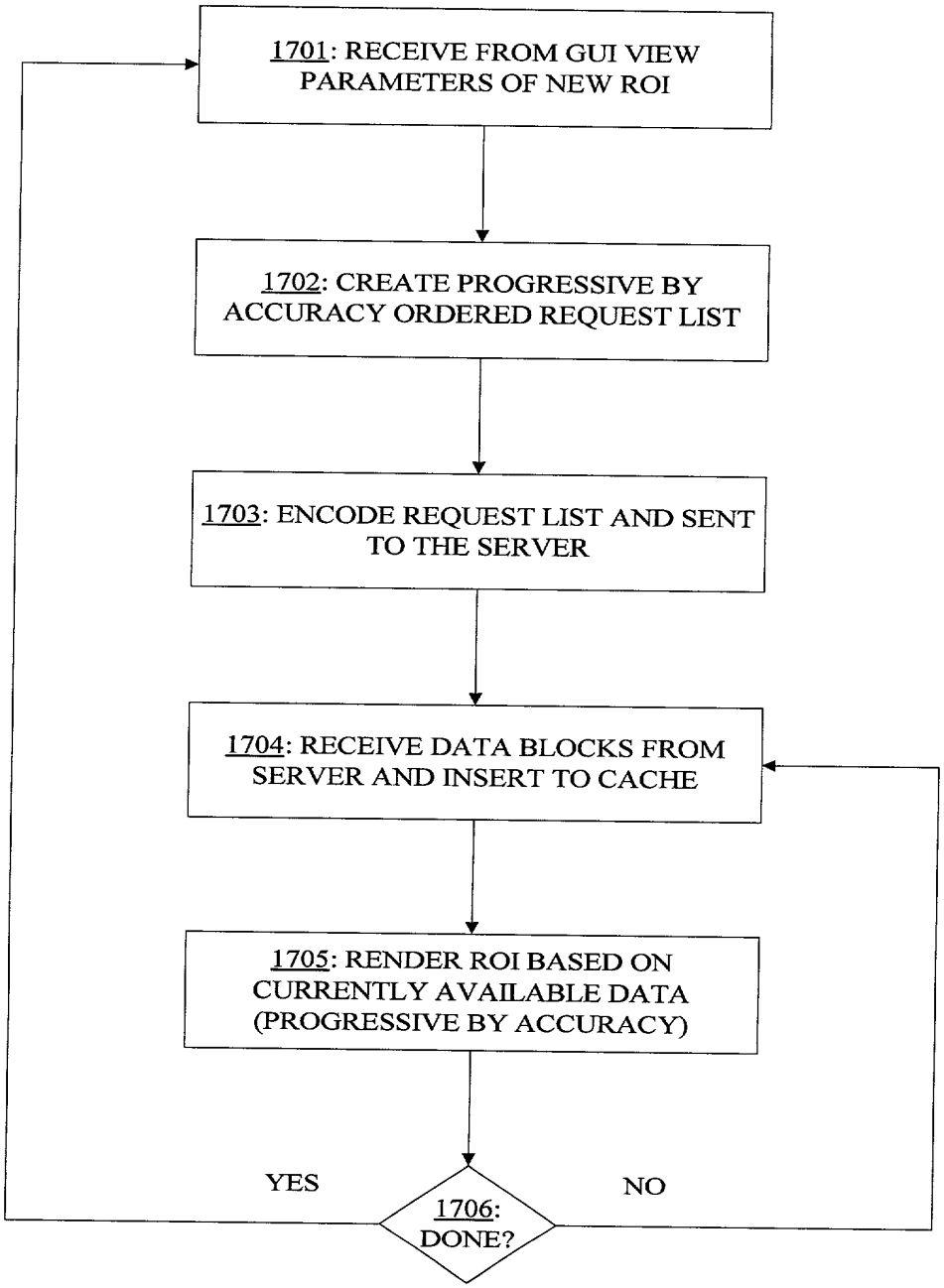

1701: RECEIVE FROM GUI VIEW PARAMETERS OF NEW ROI
1702: CREATE PROGRESSIVE BY ACCURACY ORDERED REQUEST LIST
1703: ENCODE REQUEST LIST AND SENT TO THE SERVER
1704: RECEIVE DATA BLOCKS FROM SERVER AND INSERT TO CACHE
1705: RENDER ROI BASED ON CURRENTLY AVAILABLE DATA (PROGRESSIVE BY ACCURACY)
1706: DONE?
YES
NO

Fig. 17

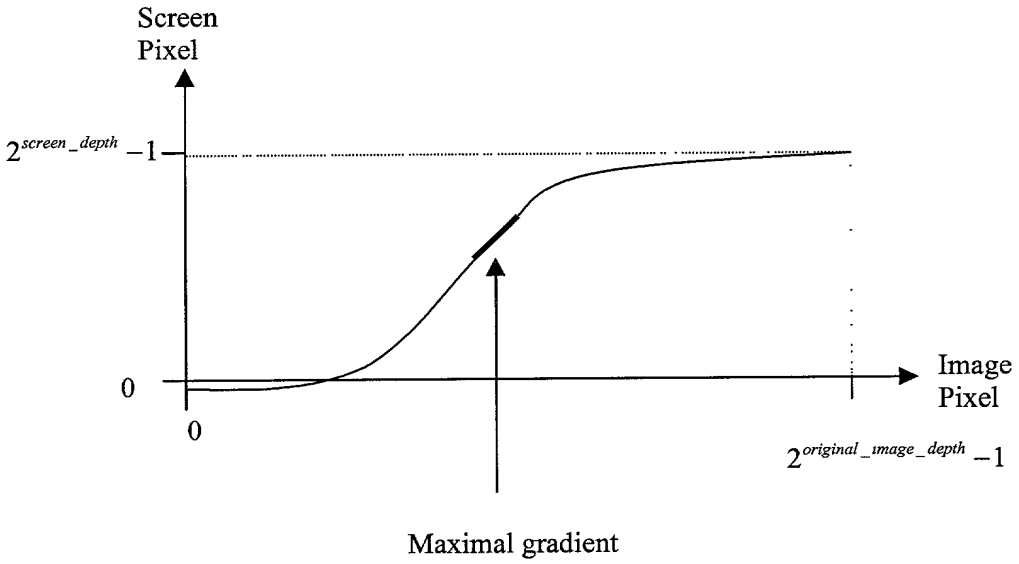

Screen
Pixel
$2^{screen\_depth}-1$
0
0
Image
Pixel
$2^{original\_image\_depth}-1$
Maximal gradient

Fig. 18

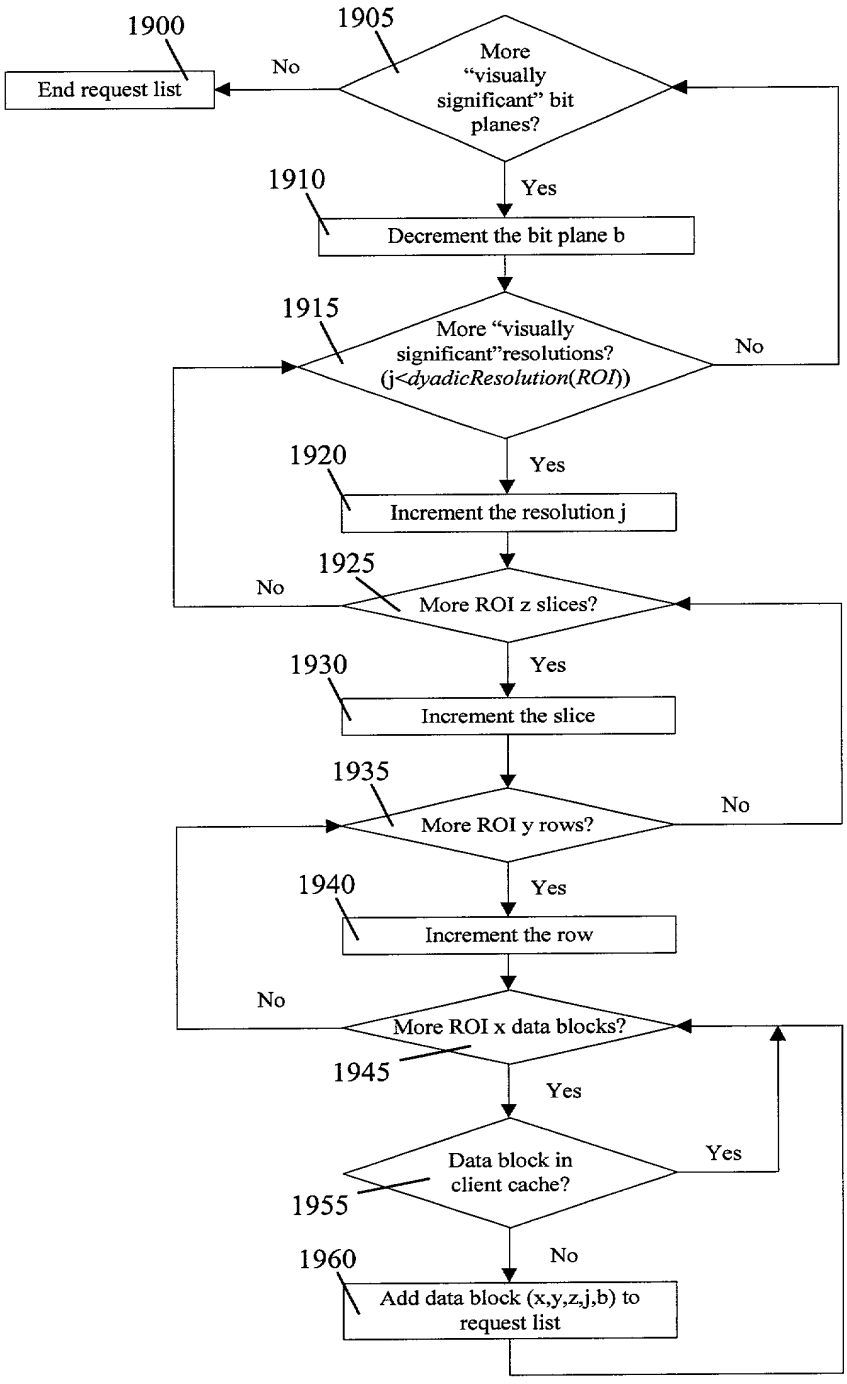

1900
End request list
1905
More "visually significant" bit planes?
No
Yes
1910
Decrement the bit plane b
1915
More "visually significant"resolutions? (j<dyadicResolution(ROI))
No
Yes
1920
Increment the resolution j
1925
More ROI z slices?
No
Yes
1930
Increment the slice
1935
More ROI y rows?
No
Yes
1940
Increment the row
More ROI x data blocks?
No
1945
Yes
Data block in client cache?
Yes
1955
No
1960
Add data block (x,y,z,j,b) to request list

Fig. 19

```
for (int res = 1 ; res <= dyadicResolution(ROI); res++) {

      for(  int z=0;
            z < NumberOfZtilesOnDyadicResolution (res,ROI);
            z++ ) {

            GetCoefficientsofLowerResolution(res, Ztile);

            for(  int x=0;
                  x < NumberOfXtilesOnDyadicResolution(res,ROI);
                  x++ ) {
                  for( int y=0;
                        y <
                        NumberOfYtilesOnDyadicResolution(res,ROI);
                        y++ ) {
                        DecodeOrExtractFromCacheSubbandCoefficients
                        ( res, x, y, z );
                  }
            }

            ExecuteInverseSubbandTransform(z);

            if( res == dyadicResolution(ROI))
                  ImageResizeAndMappingTo8bitScreen();
      }
```

Fig. 20

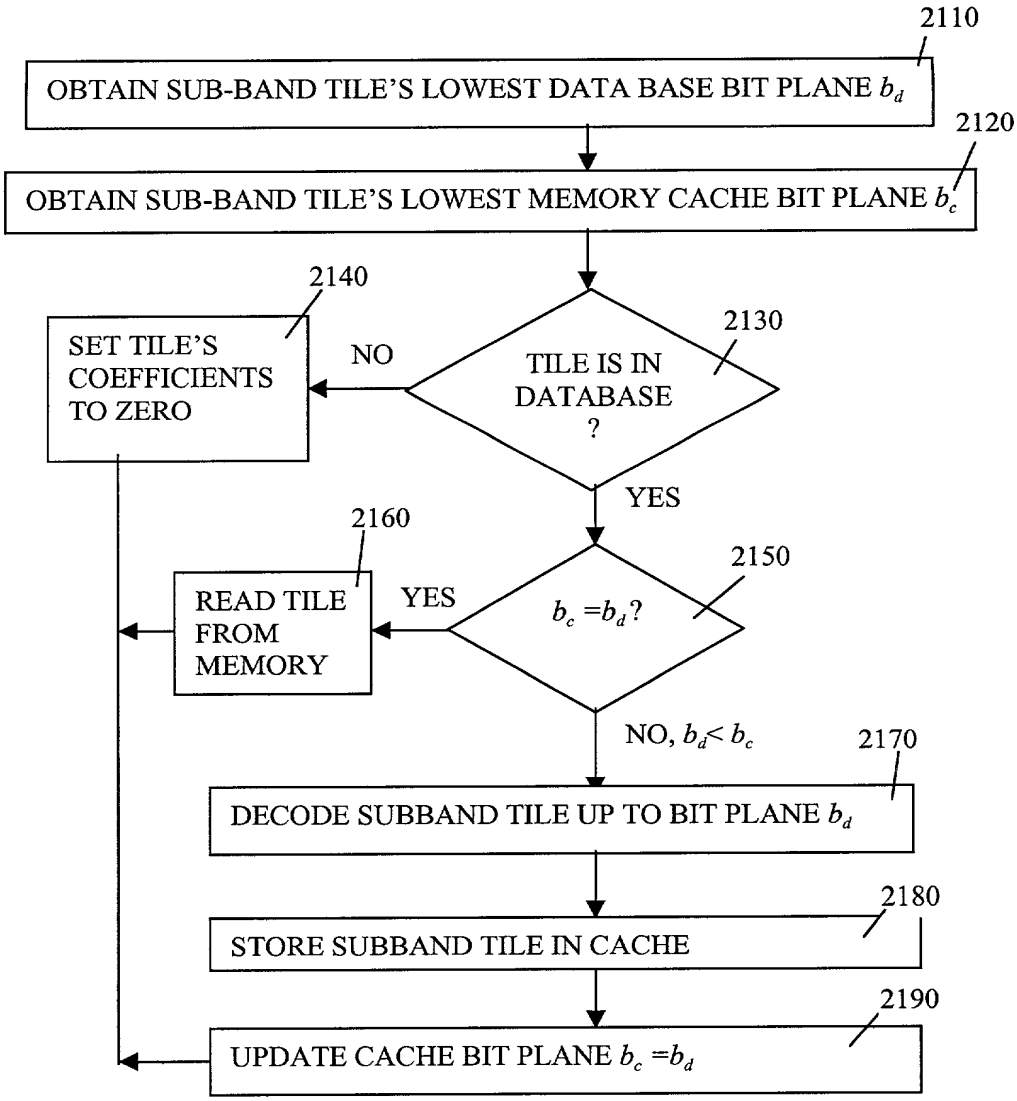

2110
OBTAIN SUB-BAND TILE'S LOWEST DATA BASE BIT PLANE b_d
2120
OBTAIN SUB-BAND TILE'S LOWEST MEMORY CACHE BIT PLANE b_c
2130
TILE IS IN DATABASE ?
NO
2140
SET TILE'S COEFFICIENTS TO ZERO
YES
2150
$b_c = b_d$?
YES
2160
READ TILE FROM MEMORY
NO, $b_d < b_c$
2170
DECODE SUBBAND TILE UP TO BIT PLANE b_d
2180
STORE SUBBAND TILE IN CACHE
2190
UPDATE CACHE BIT PLANE $b_c = b_d$

Fig. 21

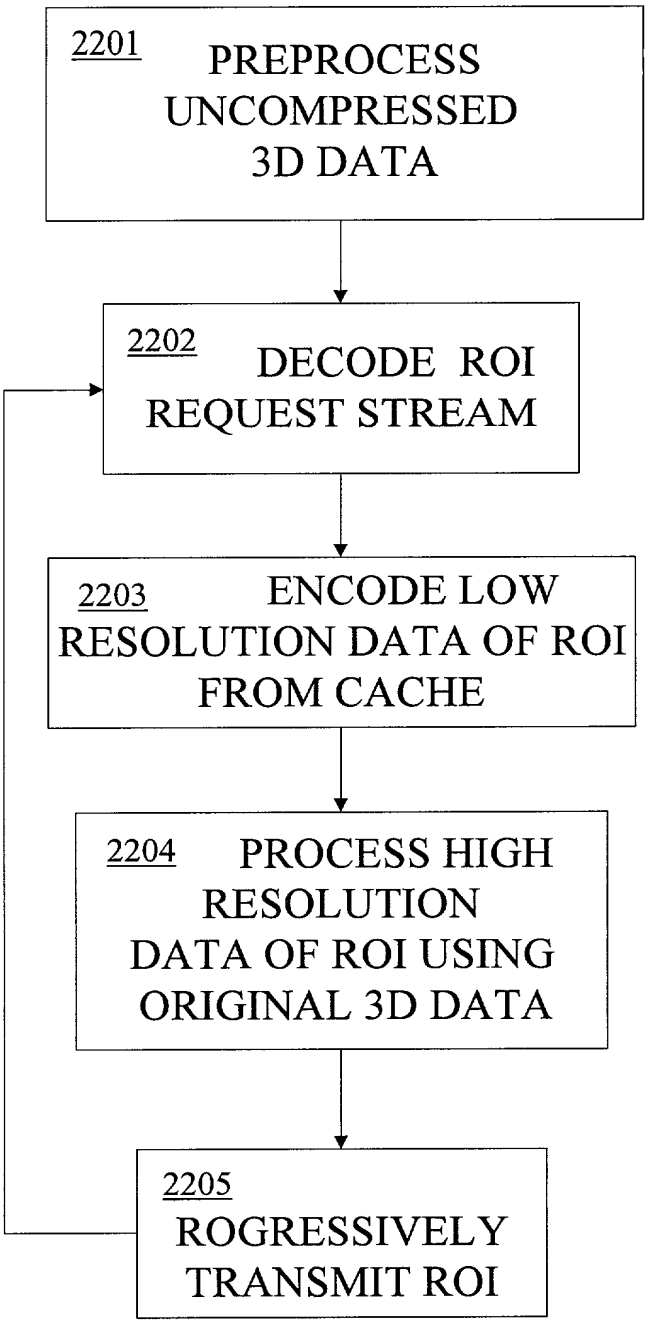

2201
PREPROCESS UNCOMPRESSED 3D DATA
2202
DECODE ROI REQUEST STREAM
2203
ENCODE LOW RESOLUTION DATA OF ROI FROM CACHE
2204
PROCESS HIGH RESOLUTION DATA OF ROI USING ORIGINAL 3D DATA
2205
ROGRESSIVELY TRANSMIT ROI

Fig. 22

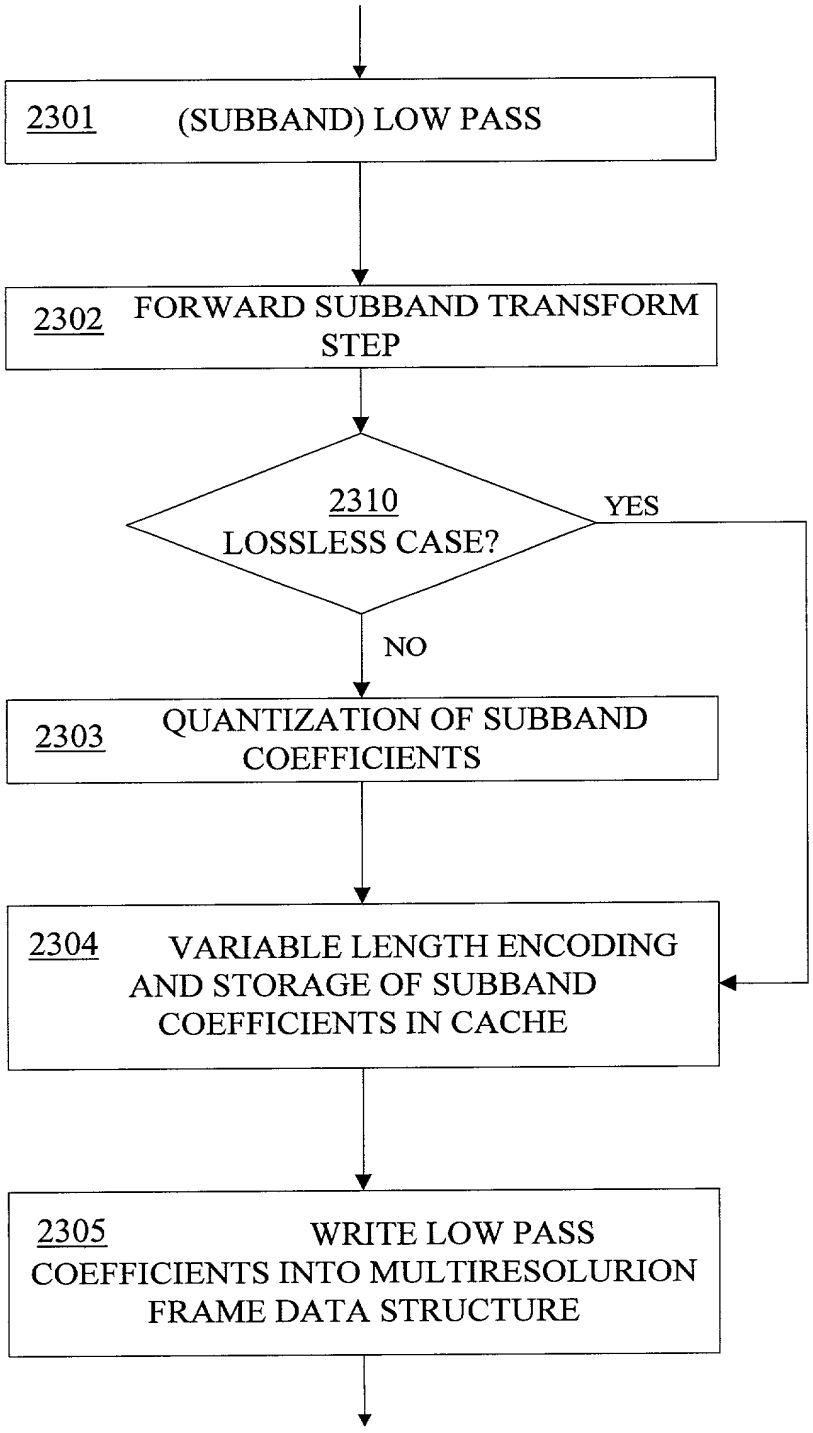

2301 (SUBBAND) LOW PASS
2302 FORWARD SUBBAND TRANSFORM STEP
2310 LOSSLESS CASE?
YES
NO
2303 QUANTIZATION OF SUBBAND COEFFICIENTS
2304 VARIABLE LENGTH ENCODING AND STORAGE OF SUBBAND COEFFICIENTS IN CACHE
2305 WRITE LOW PASS COEFFICIENTS INTO MULTIRESOLURION FRAME DATA STRUCTURE

Fig. 23

```
for(int t_Resolution=numberOfResolutions-jumpSize; t_Resolution>=1 ;
t_Resolution--) {
      leftTilesZInMemoryBuffer(t_Resolution)=
                  NumberOfTilesZInFrameMemoryBuffer(t_Resolution);
      currentTile(t_Resolution)=0;
}

for(t_Resolution=numberOfResolutions-jumpSize; ;) {
      // calculate the Z and it's resolution
      if (currentTile(t_Resolution)< nTileZ(t_Resolution)) {
          for (int t_y = 0 ; t_y < nTileY(t_Resolution); t_y++)
          for (int t_x = 0 ; t_x < nTileX(t_Resolution); t_x++)
                preprocessSubbandTile(t_x,t_y,
currentTile(t_Resolution), t_Resolution);
      }

      // update the indeces
      leftTilesZInMemoryBuffer(t_Resolution)--;
      currentTile(t_Resolution)++;

      if(currentTile(t_Resolution) < nTileZ(t_Resolution)) {
            // switch the resolution
            if(leftTilesZInMemoryBuffer(t_Resolution)==0) {
                  leftTilesZInMemoryBuffer(t_Resolution) =
                        NumberOfTilesZInFrameMemoryBuffer(t_Resolution
                  );
                  t_Resolution --;
            }
            else
                  t_Resolution = numberOfResolutions-jumpSize;
      }
      else {
            t_Resolution --;
            if(t_Resolution    1)
```

Fig. 24

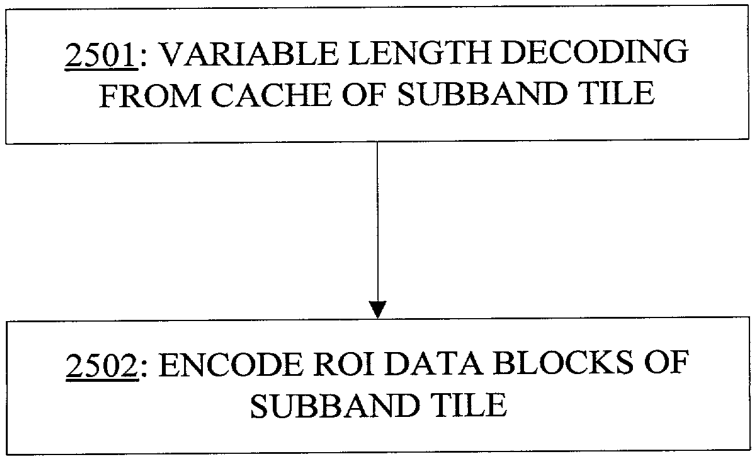

2501: VARIABLE LENGTH DECODING FROM CACHE OF SUBBAND TILE
2502: ENCODE ROI DATA BLOCKS OF SUBBAND TILE

Fig. 25

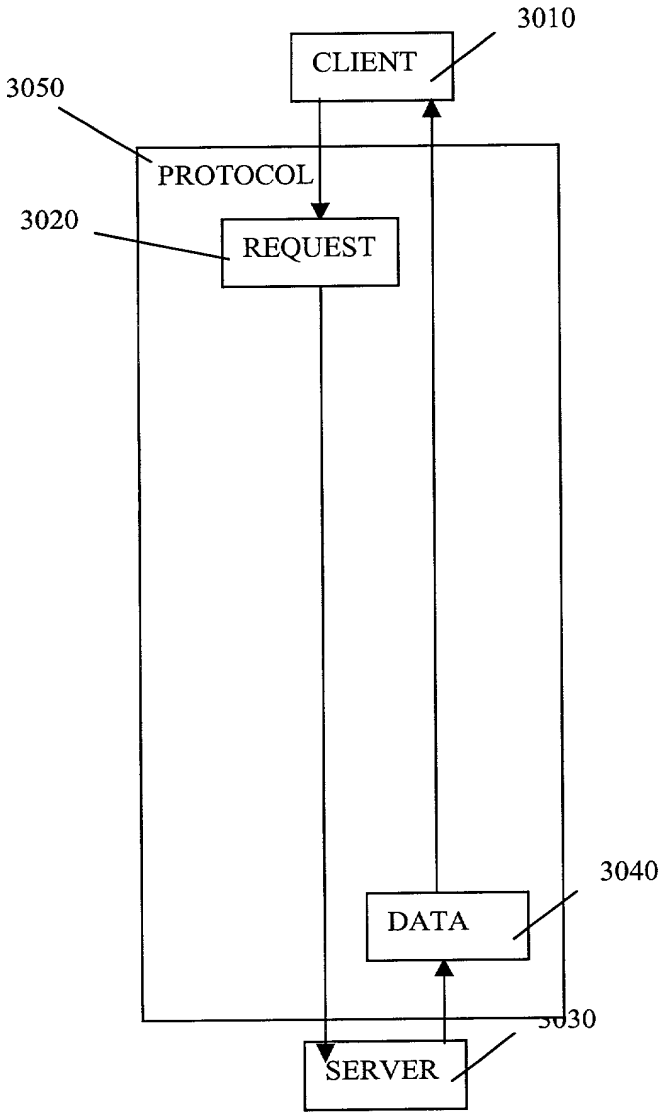

3010
CLIENT
3050
PROTOCOL
3020
REQUEST
3040
DATA
3030
SERVER

Fig. 26

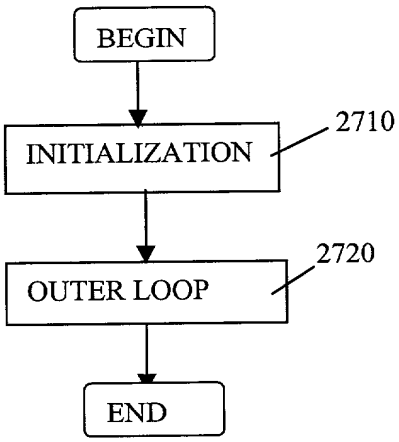


Fig. 27

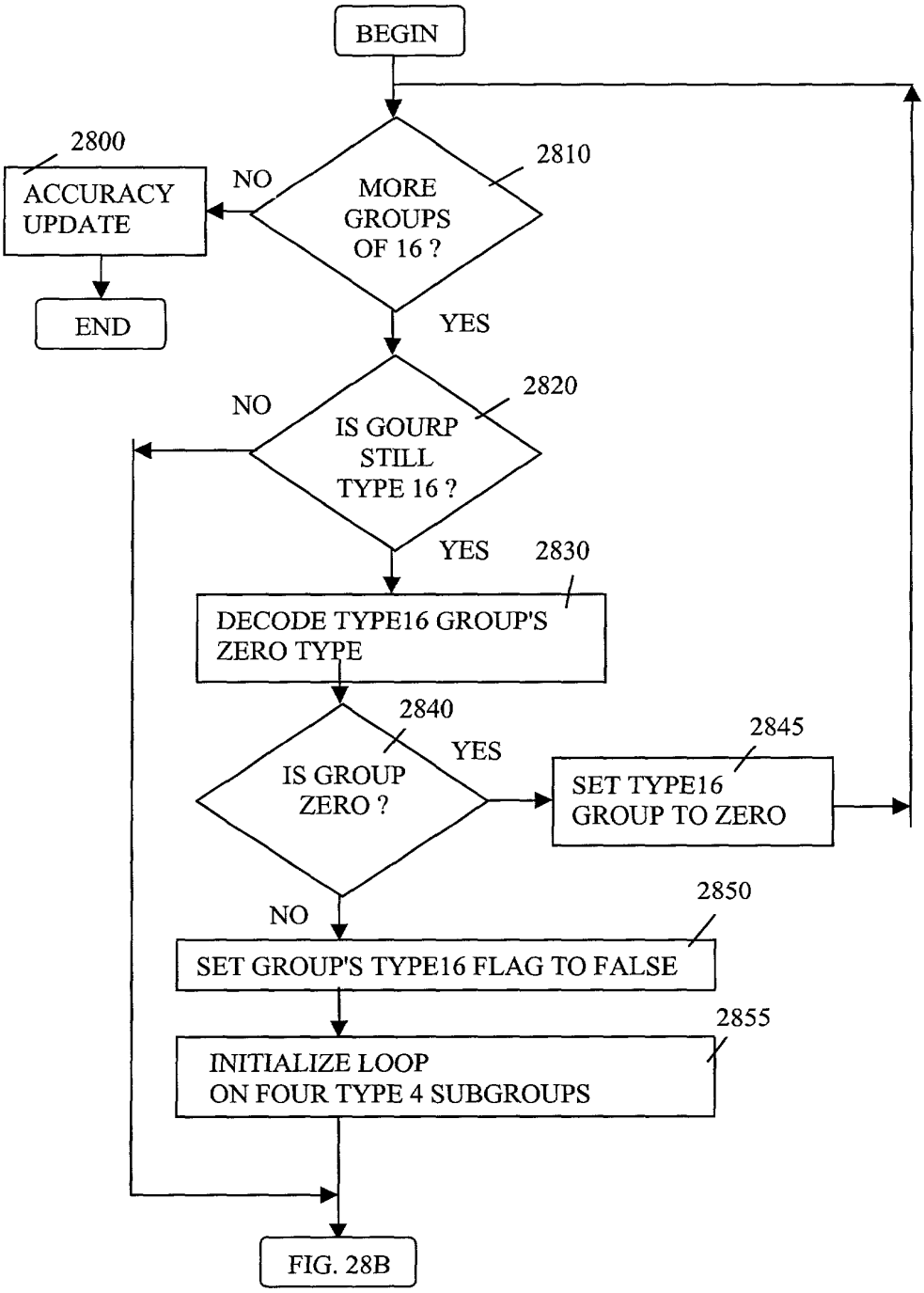

BEGIN
2800
ACCURACY UPDATE
NO
2810
MORE GROUPS OF 16 ?
END
YES
NO
2820
IS GOURP STILL TYPE 16 ?
YES
2830
DECODE TYPE16 GROUP'S ZERO TYPE
2840
IS GROUP ZERO ?
YES
2845
SET TYPE16 GROUP TO ZERO
NO
2850
SET GROUP'S TYPE16 FLAG TO FALSE
2855
INITIALIZE LOOP ON FOUR TYPE 4 SUBGROUPS
FIG. 28B

Fig. 28A

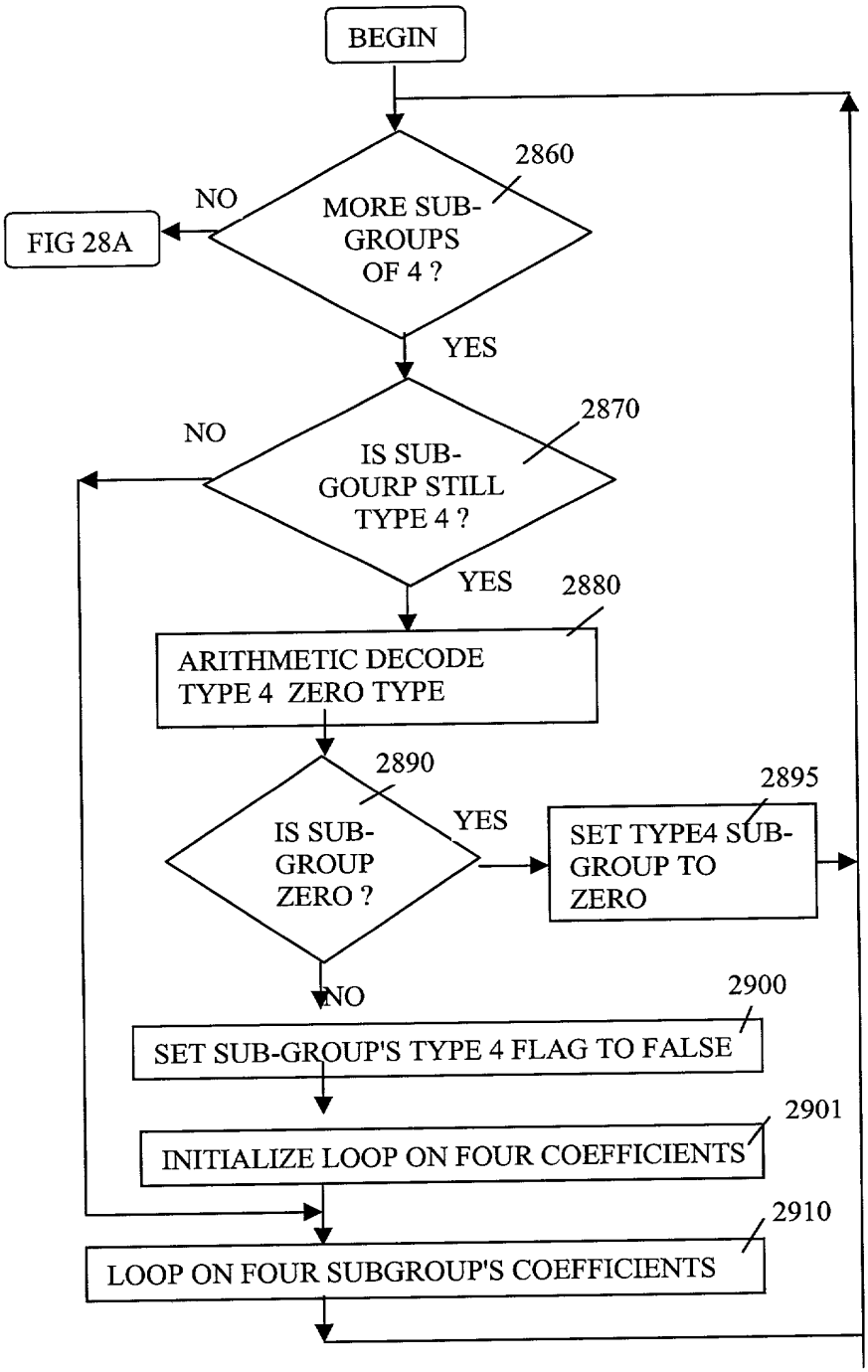

BEGIN
2860
MORE SUB-GROUPS OF 4 ?
NO
FIG 28A
YES
2870
IS SUB-GOURP STILL TYPE 4 ?
NO
YES
2880
ARITHMETIC DECODE TYPE 4 ZERO TYPE
2890
IS SUB-GROUP ZERO ?
YES
2895
SET TYPE4 SUB-GROUP TO ZERO
NO
2900
SET SUB-GROUP'S TYPE 4 FLAG TO FALSE
2901
INITIALIZE LOOP ON FOUR COEFFICIENTS
2910
LOOP ON FOUR SUBGROUP'S COEFFICIENTS

Fig. 28B

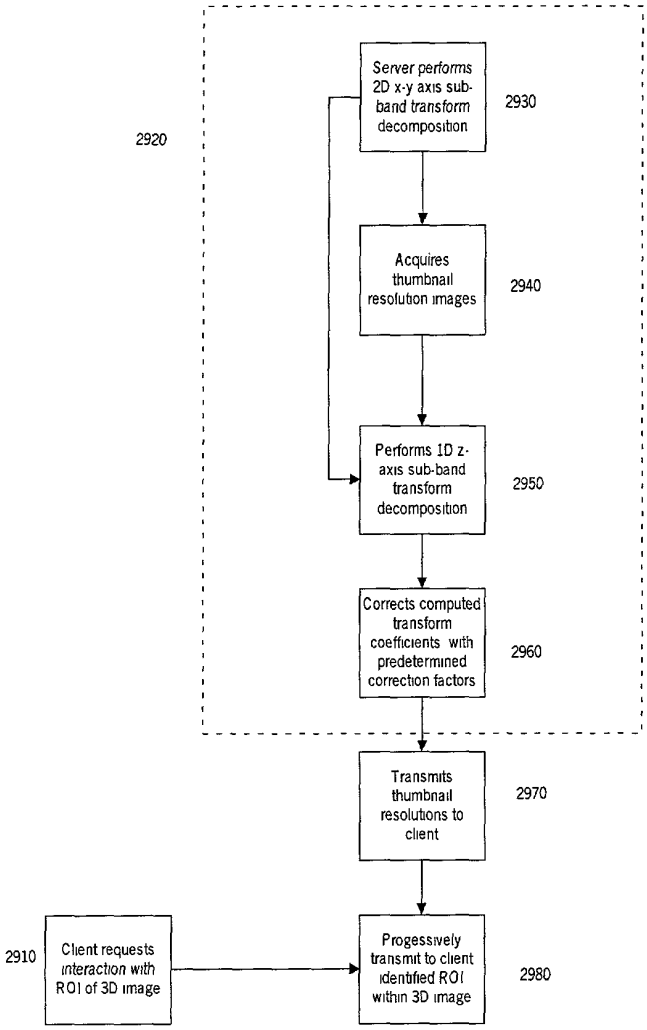

2920
Server performs 2D x-y axis sub-band transform decomposition
2930
Acquires thumbnail resolution images
2940
Performs 1D z-axis sub-band transform decomposition
2950
Corrects computed transform coefficients with predetermined correction factors
2960
Transmits thumbnail resolutions to client
2970
2910
Client requests interaction with ROI of 3D image
Progessively transmit to client identified ROI within 3D image
2980

Fig. 29

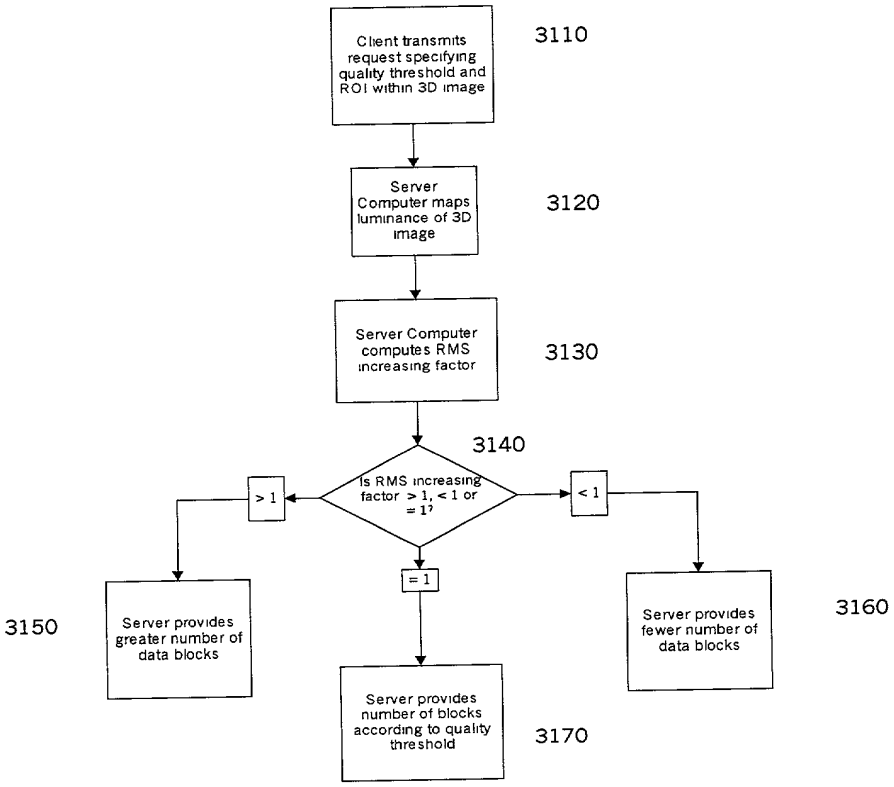

Client transmits request specifying quality threshold and ROI within 3D image
3110
Server Computer maps luminance of 3D image
3120
Server Computer computes RMS increasing factor
3130
3140
Is RMS increasing factor > 1, < 1 or = 1?
> 1
< 1
= 1
3150
Server provides greater number of data blocks
Server provides fewer number of data blocks
3160
Server provides number of blocks according to quality threshold
3170

Fig. 30